HP Availability Stats and Performance

An availability-monitoring infrastructure for HP NonStop Servers

Table of contents

Abstract

When service-level availability is monitored and coupled with specific goals and actions, the actual availability of system and application services is significantly improved. In order to achieve and maintain continuous service availability levels both operations management and business analysts benefit from goal-orientated, objectives-based monitoring of both computer system components as well as abstract application domains. Automated actions coupled with service-level goals eliminate the need for hands-on operator actions for routine exception/recovery handling and thus provides actual availability improvement. It is not sufficient to only provide high availability hardware and operating system software. Continuous enterprise service levels can only be assured if application domains and metrics are also monitored and acted upon. When availability is inadequately monitored, degraded service-levels go undetected and decreased availability or total unavailability results[1].

[1] TROM—Realization of Operations Management—ITUG 2000 – D. W. Buck

Availability

It is important to recognize that the notion of service-level availability is dependent on a user's point of view. As a result, availability-monitoring software such as HP NonStop Availability Stats and Performance (ASAP) provides users with the ability to customize their definition of availability.

Point-of-view

One view of availability is the centrally administrated view. In this view, there is one central "policy" that governs the set of availability objective rules. This type of policy is seen in "glass-room" environments where operations management personnel dictate the definition of availability.

However, availability can also be a function of each individual user's point-of-view. For example, if you are at the head of a line, your view of service availability is quite different than someone who is at the back of a line with 50 people in it. There are other important ways that the definition of availability depends on the user point-of-view. For example, a financial analyst monitoring the availability of a funds transfer application has a totally different notion of availability criteria than an operator who is worried about the state of physical objects such as controller paths and mirrored disk drives.

In a large enterprise, the number of business analysts who require "point-of-view" availability monitoring can easily out-number the people in the operations management staff. Thus "point-of-view" availability represents an important class of availability monitoring. There can be dozens of analysts at workstations throughout a large enterprise who are monitoring "availability". Each of these users can have an entirely different view of availability based on their own definition. Thus, different users must be able to customize their definition of object availability.

If graphical-user-interface availability displays are to be useful, it must be possible to segregate information. For example, a financial analyst may need to know whether the total dollar amount received in the last hour met expectations, while another user may need to know how many tickets between two cities were sold. Most often neither care about each other's notion of availability and they don't typically want their color-coded alert displays to be cluttered with alert information based on the other person's notion of availability.

Customization of availability must also provide inclusion/exclusion of properties that may or may not be used to determine availability. For example, many users may care whether a certain application process is running, but only a few may care whether the application runs in CPU 15.

In order to address such requirements, ASAP provides a range of customization options that define which entities and attributes are analyzed, as well as the state determination rules used to determine availability. These properties and rules can be customized in ASAP and thus can be used to control the state propagation engine.

If the general notion of availability were a fixed concept, state determination and propagation would be a relatively straightforward process. However, different users can and do have different definitions of service-level availability. In fact, a user's definition of availability changes many times during the day. The ASAP Client, Server, and Extension architecture is designed to operate with these requirements in mind.

State Determination

We saw that there cannot be a single definition of availability for all users. Furthermore, state determination rules can and do change for a single user in a matter of seconds. One instant a user may view availability in terms of whether the Execution Priority and preferred CPU number for application processes meet their objectives. The next minute that same user may not want these attributes factored into state determination at all, but rather may need to view state determination for processes based on busy utilization and queue lengths.

Since the definition of availability is not fixed, it follows that object state determination rules must be variable. Thus ASAP must be able to instantly change its state determination rules based on a user's relative notion of availability at any given moment. One instant using one set of attributes and goals, and at another instant include or exclude a different set of object attributes. These capabilities must be provided for both centrally administered policies as well as for customized user availability definitions.

As a result of these requirements ASAP provides customized state determination based on: 1) Central Policy Database objectives as described above; or 2) customized user defined state determination rules, or 3) combinations of both (1) and (2).

Using these customization techniques object classes such as the Process entity can have the state determination for its Priority attribute be based on centrally administered policy, but have the state of its Busy attribute be based on an individual user's customized thresholds. The computation of the overall state of the object is computed based on combinations of selected attributes and by analyzing unique state determination rules assigned to each entity attribute.

ASAP provides the following types of object-attribute state determination rules:

- Allow inclusion/exclusion of attributes for the determination of an object's state
- Use centrally administered objectives for an attribute's state determination
- Use custom user specified thresholds for an attribute's state determination
- Use application assigned state for an attribute's state determination

Since the definition of availability is dependent upon a user's point-of-view, ASAP is designed to:
1) Allow users to define the rules that determine the state of each object attribute,
2) Integrate attribute states to determine the overall state of each object, and
3) Propagate object state information upward through object class hierarchies.

State Propagation

ASAP propagates object states upward through object class hierarchies. To do this, ASAP analyzes each object's attributes, its state determination rules, its metric values, and compares those values with upper/lower bound service-level goals or objectives. As each attribute is analyzed, it is assigned an availability vector or state, for example: OK or Warning. When all the attributes for an object have been analyzed, ASAP makes an overall statement to the state of the object. Once the state of an object is determined, it is propagated upward through the object class hierarchy.

For example, if a device in the Chicago Loop named ATM\Chicago\Loop has insufficient funds and the lower bound objective for that property is not met, the state of that object can be said to be in a 'warning state'. As the state is propagated to higher levels of the object hierarchy, the status of all ATMs in Chicago inherits the 'warning state', for example, ATM\Chicago*. Finally, as the state is propagated to all ATM levels, ATMs in general can be said to be in a Warning state, for example, ATM*. This overall process is called state propagation and is displayed in a variety of ways in the ASAP Object Integration Layer (OIL)[2], the Open Enterprise Management (OEM) gateway[3], and optionally in various enterprise management frameworks such as HP Operations (formerly HP OpenView).

[2] Method and Apparatus for the Object Integration Layer (OIL) Invention disclosure—August 8, 1995—Miller
[3] Open Enterprise Management Gateway (OEM) Invention disclosure—October 9, 2001—Miller, O'Brien, Tremblay

Product Overview

HP ASAP software provides an infrastructure for monitoring the availability and performance of system, subsystem, and application objects on HP NonStop Servers[4]. Both availability stats and performance information are integrated to form normalized availability vectors for monitored domains and associated properties. Information integration includes operational status, performance, and availability objectives for HP NonStop server systems, subsystems, and abstract application domains. Additionally, ASAP provides automated operations capabilities coupled with service-level goals to eliminate the need for hands-on operator-actions for routine exception and recovery handling allowing ASAP to directly improve actual enterprise availability.

ASAP implementation consists of four functional areas: Client, Server, Extension, and Hybrid.

ASAP Client

The ASAP Client provides one means of visualization and analysis of object-state information[5]. Other forms include enterprise management frameworks, EMS events, Web Viewpoint, HP Systems Insight Manager, HP Operations (OpenView), SNMP Traps, HTML Email, wireless phone and pager notifications, as well as structured database query. ASAP Client communicates with the ASAP Server and Extension components running on HP NonStop Kernel. The ASAP Client provides:

- A graphical user interface with real-time, state-propagating, and hierarchical views
- 3D, color-coded, and availability-vector graphs with context sensitive drill-down reporting
- State change scoreboards and logs with context sensitive historical drill-down reporting
- Late/early statistics validation for the detection of non-responding objects/domains
- Full customization of monitored entities and availability vectors defining object availability
- Historical object cache and download Wizard for historical analysis of availability data
- Interactive Development Environment for the development of new entity definitions
- Rich graphical three-dimensional interface that executes on Microsoft Windows
- Intelligent-Agent functions that allow forwarding/sharing of object-state information with enterprise management frameworks.
- Automatic alerts and reports sent to wireless phone, pager, and email.
- Notification through wireless phone, pager, email on state change or at various times of the day.
- Automatically generated HTML or EXCEL data for inclusion in Web sites or management reports.

ASAP Server

The ASAP Server executes on HP NonStop Kernel. The ASAP Server monitors, gathers, and analyzes availability, state, and performance statistics throughout a network of HP NonStop Kernel (NSK) clusters and optionally takes automated corrective actions. The ASAP Server provides the following features:

- Statistics Gathering Processes (SGPs) collect availability and performance information
- NonStop ASAP Monitor process pairs maintain persistent NonStop SGPs on each node
- SGPs are designed to collect statistics with extremely low performance overhead
- SGPs forward object-state information to NonStop ASAP database collectors
- ASAP Collectors store availability data in a third-normal-form, real-time, and historical database
- ASAP Servers can be configured to operate autonomously on a single node

[4] ASAP SE30v2—Release June 2001—Service-level objectives and support for: CPU, Disk, Expand, File, Node, Process, ProcessBusy, RDF, Spooler, System, Tape, and TMF.
[5] Asap Downloads—http://www.NonStopASAP.com

- Servers can operate collaboratively to form super-clusters of availability vector information
- Optional inter-node time-of-day synchronization provided between monitored nodes
- ASAP monitors availability and performance of systems, subsystems, and applications
- System objects—include CIP, CPU, Disk, Expand, Expand/IP, Node, System, and Tape
- Subsystems—include File, Sub-volume, Process, TCP/IP, Comm, RDF, Spooler, and TMF
- Applications—can be externally monitored using the Process entity
- Application Domains—can be internally monitored using the ASAP Extension (ASAPX) Application Programmer Interface (API)
- Selection of monitored objects can be either through auto-discovery or manual object selection
- User defined upper-and-lower bound GOAL threshold monitoring is provided.
- GOALs can be set on classes, sub domains, individual objects, and hierarchical combinations
- GOALs are permanently remembered in a fully audited objectives database[6]
- GOALs can optionally generate traps or EMS events
- ACTIONs can optionally take automatic corrective actions, such as suspend a process, defragment a disk, or secure files that are improperly secured
- ASAP Server provides interfaces to the ASAP Extension API, allowing customer applications and third party plug-ins to benefit from the ASAP Client/Server infrastructure when application programs use the optional ASAPX product
- ASAP File SGP monitors both Guardian and OSS files
- File SGP has the ability to group files into logical and hierarchical groups
- ASAP Process SGP monitors both NSK processes as well as OSS processes
- Process SGP has the ability to group files into logical, hierarchical groups, and to compute aggregate statistics for the entire abstract group
- Service-level objectives may be set on both individual objects as well as aggregate groups
- Automated actions can be associated with service goals, allowing ASAP to perform corrective automated operations

ASAP Extension

The ASAP Extension product (ASAPX) executes on the HP NonStop Kernel[7]. The ASAPX product provides an API into the ASAP infrastructure so that the availability and performance of abstract application domains can be monitored. The ASAP Extension:

- Allows application domain statistics to be integrated with ASAP Client/Server infrastructure
- Application programs benefit from the same ASAP architecture used for HP objects
- Collects statistics through an ultra-high-performance and shared-memory architecture
- Provides measurement, viewing, and analysis of application service-level objectives
- Automatically evaluates pre-defined objectives to establish alert priorities
- Tracks the productivity, performance, and availability of applications
- Notifies operators and administrators when application processes do not meet objectives
- Notifications occur through fat/thin clients, EMS, wireless phones, pagers, and HTML email

[6] ASAP Objectives database can be TMF protected, but TMF is not mandatory
[7] ASAPX Extension—Optional product SE31v2

ASAP Hybrid

ASAP Hybrid extends the capabilities of ASAP application monitoring to remote systems running the Linux operating system. Application metrics from the remote Linux systems are seamlessly integrated into ASAP on one or more NonStop servers, thereby providing an overall view of the entire application as it spans one or more NonStop servers and one or more Linux servers. ASAP Hybrid:

- Provides a Linux API largely identical to that furnished on the NonStop server
- Employs an ultra-high-performance, shared memory, and non-blocking mechanism for collecting application metrics on Linux
- Integrates fully with the ASAP framework, allowing users to set objectives, generate alerts, and take actions based on Linux application data
- Detects if a Linux system hosting an instrumented application fails and generates corresponding alerts
- Supports all external ASAP interfaces, including thin/thick clients, EMS, HP Systems Insight Manager (SIM), HP Operations (OpenView), email, and wireless phone/pager interfaces.

ASAP Interface Summary

Rather than focus on a single enterprise management framework or a single interface technology, ASAP provides numerous interfaces so it can be integrated into an extremely wide range of computing environments.

EMS Tokenized Event Interface: All ASAP alerts, goals, actions can be optionally reported through NonStop tokenized EMS (Event Management Subsystem) event interface.

Provider APIs: Customer and third party applications can add their own entities through ASAP's simple-to-use API interface. This API extends the same alert, goal, and action features to custom entities that are provided to system and subsystem entities that ship with the standard ASAP core product.

Consumer APIs: ASAP includes the HP Open Enterprise Management (OEM) gateway allowing customized features and/or analysis to be developed. Both traditional plain text and XML interfaces are provided, as well as standard plug-in interfaces to enterprise management frameworks such TNG and HP Operations (OpenView).

Application/Third Party plug-ins: Application/Third Party plug-in API interface allows new customer and third party entities to be added to the ASAP environment, allowing all ASAP features to be uniformly extended to system, subsystem, and application entities.

Published Database: The availability, state, and performance ASAP database is a fully published third-normal form database, allowing public batch query of objects, states, and statistics.

Graphical User interface: ASAP includes rich three-dimensional thick client interfaces, with color-highlighted, icon-guided, hot-spot drill-down. ASAP also includes a wide variety of other user interfaces including thick, thin, command line, enterprise management, and notification alert/report interfaces.

Conversational interface: ASAP includes an extensive conversational command line interface, allowing batch scripts, configuration, obey form, and archival of command streams. ASAP also includes a wide variety of other user interfaces including thick, thin, enterprise management, and notification interfaces.

Notification Alert-Report interface: ASAP includes a wide range of notification interfaces, including SMTP email, graphical HTML email, wireless phone, pager, SMS, XML, and HTML web page services.

Optional enterprise management frameworks (**EMF) interfaces:** ASAP includes optional interfaces to EMFs such as HP Systems Insight Manager or HP Operations (OpenView).

HP SIM and Operations (OpenView): ASAP includes built-in interfaces to HP Systems Insight Manager and HP Operations (OpenView) for no additional charge. While these interfaces are built-into ASAP, neither SIM nor OpenView are required in order to use or deploy HP ASAP. Thus ASAP can be used with or without SIM or OpenView.

Web Viewpoint Plug-in: The Web ASAP plug-in (WASAP) for Web Viewpoint provides a thin-client interface to ASAP state and statistic information through the HP Web Viewpoint product.

Thin client interfaces: Built into ASAP allow local/proxy Web site hosting in the core product.

In addition to the functional layers and interfaces described above, ASAP includes additional architectural features discussed below.

Database

ASAP includes a database that encapsulates both statistical and service-level objective information. Statistical information includes availability, statistics, and performance data. Objectives information includes user-specification of which objects should be monitored and the service-level objectives for monitored objects.

Statistics Data

The ASAP statistical database contains both current and historical availability, stats, and performance information. The database is self-maintaining; it can be configured to retain various amounts of data based on user-defined options. It can retain information indefinitely; purge data daily at midnight; retain varying amounts of history from one day to 10 years for each type of monitored object; or rollover database files at a pre-configured time of day. Since the database contains both current and historical availability data, the database is used for both online and historical reporting.

The ASAP database schema is published at http://docs.hp.com and thus is available for access by customers and third parties for real-time or batch-query statistical analysis[8]. Since the database is normalized in third normal-form and because it can be fully partitioned, the ASAP database lends itself to high-performance super-scaling for applications where continuous online monitoring and archival of high-availability information is a requirement.

Objectives Data

In addition to statistical data, the ASAP database also contains user-configured object-selection and availability-objectives criteria. Objectives are stored in a fully audited and recoverable database.

The selection of objects, their service-level objective settings, and any automated actions are permanently retained, even after CPU reloads, or even in the extraordinary event of a catastrophic system failure. Service-level objectives contained in the database also support both upper and lower bound specifications, For example, GOAL Process, Busy>10, Busy<60; as well as multiple property objectives, e.g.

```
GOAL Process $XYZ, Busy>10, Busy<60, QLen<7, Pri=150
```

Database objectives also permit both individual domain objectives as well as hierarchical specification. For example, you can specify all disks should be less than 60% busy, but certain disks such as $Data1 and $Data2 should be treated differently, for example less than 30% busy.

```
GOAL Disk, Busy < 60       ! All other disks should be less than 60% busy
GOAL Disk $Data1, Busy < 30 ! $Data1 should be less than 30% busy
GOAL Disk $Data2, Busy < 30 ! $Data2 should be less than 30% busy
```

[8] ASPDDLDB – DB Schema published for customer and third party access

Entity Definition Language

ASAP Client, Server, Extension, and Hybrid components incorporate an Entity Definition Language (EDL) that provides extensible definition of abstract entities and attributes as they relate to ASAP features and functions. EDL allows system entities, customer application domains, and third party entities to be defined externally to the ASAP environment.

Metadata Namespace

All entities and attributes monitored by ASAP are defined using the EDL language. The entire ASAP service-level objectives namespace is also defined by EDL. Each entity that is defined to ASAP using EDL is an entity whose availability, stats, and performance can be monitored.

Entities, Attributes, and Data

The notion of Entity and Attribute in ASAP are somewhat synonymous with the notion of table and column in the SQL data model. An Entity can be thought of as a table. An Attribute can be thought of as a column in a given table. ASAP differs from the SQL model in that entities and attributes have intrinsic properties that relate specifically to ASAP features and functions. EDL also allows data to be included in an EDL file, so an EDL file can represent the encapsulation of entity-attribute schema, statistics, and state information for running system(s)[9].

Interactive Development Environment

The ASAP Client includes an interactive development environment (IDE) that is used by software developers to interactively develop system/application entity definitions. The IDE includes context-sensitive interactive help for the EDL language. The IDE includes functions that allow editing, compiling, exporting, and importing EDL environments. The EDL IDE also allows interactive saving of data from live host sessions along with entity definitions.

ASAP Extension and Hybrid

EDL also defines entities and attributes monitored by the ASAP Extension and Hybrid components on behalf of application entities. This includes how ASAPX defines and treats application data in shared memory, how it computes the values of individual attributes, as well as the format of the record produced for an application in the ASAP database.

Sharing

On Microsoft Windows, EDL files are text files with an "EDL" file extension. An EDL file contains one or more EDL statements and thus the file provides a portable container representing an ASAP environment.

Users and developers can share EDL "environments" with other users/workstations. Copying or mailing an EDL file to another workstation provides sharing of schema, statistics, and state information. Since EDL is a registered extension in MS-Windows, double-clicking an EDL file, for example, within Outlook, re-displays the original environment that created the EDL.

Capturing Environments

EDL files can contain data representing availability, statistics, and state information for a running environment. For example, when an ASAP Client is monitoring system and application objects; the schema, statistics, and state information for the entire environment can be saved to an EDL file simply by clicking the "save" toolbar button.

[9] EDL Specification—Section 6—Asap2 Client TIM manual

Capture of entire system and application environments using EDL is beneficial because it:

- Provides portable encapsulation of application and system entity definitions.
- Allows entities, attributes, states, and statistics to be stored in a single EDL file.
- Allows different sets of customized ASAP settings.
- Allows environments (and data) to be shared.

Benefits

EDL allows the capture and sharing of environments that represent observed system and application behavior. This capability has been shown to be useful for:

Problem Reporting: Captures environments, including statistics and state information

Prototyping: System or Application Entity, Attribute, Data

Collaboration: Group discussion of observed behaviors

Education: Demo features of an environment

QA/Testing: Reproducing test scenarios

Monitoring

Consistent with continuous availability requirements, ASAP allows dynamic selection of monitored objects. Monitored objects can be added or removed while the ASAP system is running. If no objects are selected for an entity class, the Statistics Gathering Process for that entity will automatically configure a set of objects. For example, if you do not specify a CPU to be monitored, all CPUs will be monitored.

Selection

The ASAP MONITOR command[10] allows specification of which objects should be monitored. Specific ASAP entities such as CPU, Comm, CIP, Disk, Expand, File, Process, Process Busy, RDF, Spooler, Tape, TCP, TMF, and more can be selectively monitored using the MONITOR command. For example, if you want to externally monitor Processes $App, $Funds, and $Atms you would use the following MONITOR commands:

```
+ MONITOR PROCESS $App
+ MONITOR PROCESS $Funds
+ MONITOR PROCESS $Atms
+ COMMIT
```

Also consistent with continuous availability requirements, objects can be removed from service at any time. For example, monitoring of process $App could either be temporarily suspended or stopped altogether with either of the following commands:

```
+ MONITOR PROCESS $App, OFF    -- suspends monitoring, keeps object in
data base
+ MONITOR PROCESS $App, DELETE -- removes object from objectives
data base
```

Commit processing

User defined objects and associated objectives are permanently saved in the ASAP objectives database. Database changes are not applied to ASAP monitoring components until you "commit" changes using the COMMIT command[11].

[10] ASAP MONITOR command—Section 5—ASAP Server manual
[11] ASAP COMMIT command—Section 5—ASAP Server manual

The COMMIT command tells ASAP SGPs to reload their objectives from the ASAP objectives database. The COMMIT processing architecture used by ASAP allows batched, instantaneous alteration of large numbers of objects in conjunction with efficient dynamic reconfiguration of objectives. This allows the entire monitoring configuration to be instantaneously and radically changed at various times of the day.

Goals and Actions

ASAP provides a rich set of service-level objective capabilities. Service-level objective features provided by ASAP are often referred to as Discrete Object Thresholds, DOTs, or simply GOALs.

The DOTs acronym is useful since it helps visualize ASAP's ability to focus on a specific dot in the huge universe of objects and properties. DOTs provide both upper and lower bound objectives, for example, Busy>30, Busy<50, as well as complex specification of availability property objectives for multiple attributes. DOTs can be set on classes, sub domains, and individual objects. DOTs are permanently retained in the audited objectives database. Automated actions can also be associated with DOTs forming overall GOAL and ACTION automated operations scenarios.

GOALS

The ASAP GOAL command[12] allows specification of discrete object thresholds for monitored objects. Any data attribute defined in ASAP that has a state pair vector can have a goal associated with it. For example, if you want the performance objective for CPUs to be less than 50% busy, and to have a Queue of less than 6 processes waiting to execute, you would use the following command:

```
+ GOAL CPU, BUSY < 50, QUEUE < 6[13]
```

Service-level objectives

The GOAL command makes a statement about the preferred service-level objectives for an object or class of objects. (The RANK command is synonymous with the GOAL command).

Examples:

```
GOAL ATM Chicago, CASH > 2000
```

Asserts the objective that ATM machines should contain more than $2000. Service-level objectives can also be defined for hierarchical domains. For example, if you would like a different objective for ATMs in the Chicago Loop, you could use the following syntax:

```
+ GOAL ATM,               CASH > 1000   -- Objective for all ATMs
+ GOAL ATM Chicago,       CASH > 2000   -- Objective for all Chicago ATMs
+ GOAL ATM Chicago\Loop, CASH > 4000   -- Objective for Chicago Loop ATMs
```

When service-level objective thresholds are not met, alerts are displayed in the following ways:

- Graphically in the ASAP Client with color-coded Icons, graphs, logs, tree, and list views.
- Graphically in built-in thin-client web-based displays
- Optionally through tokenized EMS subsystem events in standard NonStop Kernel EMS logs.
- Graphically in the Open Enterprise Management gateway.
- Graphically in participating Enterprise Management Frameworks.
- Through graphical HTML email, wireless phone/pager, or short SMS text messages.

[12] ASAP GOAL command—Section 5—ASAP Server manual
[13] Commit command applies Monitor/Rank objective changes

Objectives examples

ASAP provides two objective commands, MONITOR and GOAL, which are used to specify object selection and service-level objectives for system objects, subsystems, and application domains. The following provides examples of possible objectives:

To monitor a DISK named $DATA, you would use the following syntax:

```
+ MONITOR DISK $DATA
```

To assert that the objective for all DISK volumes should be less than 75% Full, you would use the following syntax:

```
+ GOAL DISK, FULL < 75
```

To assert that the objective for a specific DISK $DATA should be less than 80% Full, you would use the following syntax:

```
+ GOAL DISK $DATA, FULL < 80
```

To set objectives for DISK volume $DATA to be less than 80% Full, to be less than 30% Busy, and to have a request queue length (QLEN) less than 5, you would use the following GOAL syntax:

```
+ GOAL DISK $DATA, FULL < 80, BUSY < 30, QLEN < 5
```

To set objectives for DISK volume $DATA to have its primary CPU be equal to 1, the Primary disk (P) controller-path be the Primary "P" path, and for the Mirror disk (M) controller-path be the Backup "B" path, the syntax would be:

```
+ GOAL DISK $DATA, CPU = 1, P = "P", M = "B"
```

To monitor the file $System.System.Userid, set the service-level availability objectives to be owned by "255,255", and to be secured with an RWEP security vector of "OOOO", the syntax would be:

```
+ MONITOR FILE $System.System.Userid
+ GOAL FILE $System.System.Userid, OWNER="255,255", RWEP="OOOO"
```

To set the service-level availability objective for all CPUs to be less than 70% busy, except for CPU 5 whose objective should be less than 55% busy, and its Queue should be less than 4; the syntax would be:

```
+ GOAL CPU,   Busy < 70
+ GOAL CPU 5, Busy < 55, Queue < 4
```

To set the service-level availability objectives for an ATM machine named "NorthMain" to have more than $2000 CASH, to specify its Transaction rate should be less than three transactions per interval, and to specify that bad pin verifications should be less than 4 per sample, the syntax would be:

```
+ GOAL ATM NorthMain, CASH > 2000, Trans < 3, BadPins < 4
```

To monitor a running process $FUNDS, assure its continuous availability, set its service-level availability objectives to be less than 25% Busy, its request queue length (QLEN) less than five requests, its primary CPU = 1, and its Priority > 150, the syntax would be:

```
+ MONITOR PROCESS $FUNDS
+ GOAL PROCESS $FUNDS, Busy < 25, QLEN < 5, Cpu = 1, Pri > 150
```

To monitor the Remote Database Facility (RDF) subsystem between NonStop clusters \Chicago and \Newyork, and set the service-level availability objective for the audit trail Relative-Time-Delay to be less than 90 seconds, the syntax would be:

```
+ MONITOR RDF Chicago->Newyork
+ GOAL RDF, RtdSecs < 90
```

Objective events

ASAP provides EMS events or "trap" generation when service-level objectives are not met.

Very fine control of event generation is provided. You can specify event generation down to a specific objective for a specific object. For example the following syntax would "repeatedly" generate "critical" EMS events if the RWEP is not "OOOO" for the system Userid file:

```
+ RANK FILE $System.System.Userid, RWEP = "OOOO" CRITICAL REPEAT
```

The following would generate an EMS event when the priority of process $XYZ was not 150, but only one event would be generated, it would be an informative event (INFO) and would not be displayed on the operator console (for example, only for programmed automation purposes).

```
+ RANK PROCESS $XYZ, Pri = 150 INFO NODISPLAY
```

ACTIONS

In addition to being able to declare service-level GOALs in ASAP, you can also optionally associate automated ACTIONs with goals. Automation can encompass a wide range of automatic actions from re-securing a file if someone is attempting to "tamper" with the system, to altering the priority of a process, to automatically compressing a disk when ASAP detects it has become overly fragmented. Below are examples of just a few types of actions that can be associated with service-level goals.

When Files do not meet security goals, ASAP can raise an alert and take a corrective action. For example, if the security vector of the $System.System.Userid file is not "OOOO" the following GOAL and ACTION statement will automatically correct such a security violation.

```
+ GOAL FILE $System.System.Userid, RWEP=OOOO ACTION SECURE
```

When Disk volumes do not meet their maximum free space fragmentation goal, ASAP can raise an alert and take a corrective action. From the example below, if the largest free space fragment for any disk volume is not greater than 100 megabytes, the following GOAL and ACTION statement will automatically generate an event and take corrective action to compress the free space fragmentation.

```
+ GOAL DISK, FRAGMENT > 100 CRITICAL ACTION "DCOM <#object>"
```

When a Disk volume becomes too full, ASAP can raise an alert and take a corrective action such as run a TACL macro. From the example below, if any disk volume becomes greater than 90% full, the following GOAL and ACTION statement will automatically generate an event and take corrective action to run a TACL macro to purge temporary files.

```
+ GOAL DISK, FULL < 90 CRITICAL ACTION "TACL CLEANUP <#object>"
```

When there are insufficient RDF audit trail extractors, ASAP can raise an alert and take a corrective action such as run a TACL macro to add extractors. From the example below, if the backup RDF relative time delay becomes greater 60 seconds, the following GOAL and ACTION statement will automatically generate an event and take corrective action to run a TACL macro to add extractors.

```
+ GOAL RDF, RTDSECS < 60 CRITICAL ACTION "TACL ADD_EXTRACTORS <#object>"
```

When a TMF transaction runs excessively long, ASAP can raise an alert and take a corrective action such as Abort the transaction or run a TACL macro to analyze the transaction and take an action. From the example below, if the TMF transaction duration runs longer than 900 seconds (15 minutes) the following GOAL and ACTION statement will automatically generate an event and take corrective action to abort the transaction or to run a TACL macro.

```
+ GOAL TMF, DURATION < 900 CRITICAL ACTION "TMFCOM ABORT <#object>"  (or)
+ GOAL TMF, DURATION < 900 CRITICAL ACTION "TACL XACT_ANALYZE <#object>"
```

When a running Process unintentionally stops/abends, ASAP can raise an alert and take a corrective action such as re-start the application process or server class. From the example below, if a process named $DBAPP stops, the following GOAL and ACTION statement will automatically generate an event and take corrective action to re-run the application.

```
+ GOAL PROCESS $DBAPP, STATUS ACTION RESTART
```

When a Process loops, becomes overly busy or has the wrong priority, ASAP can raise an alert and take a corrective action to suspend or lower the priority of the process. From the example below, if CPU Busy for any object file exceeds 50% busy or the priority is incorrect, the following GOAL and ACTION statement will automatically generate an event and take corrective action to suspend or lower the priority of the application.

```
+ GOAL PROCESS $Data.Apps.DBapp, BUSY < 50 ACTION SUSPEND            (or)

+ GOAL PROCESS $Data.Apps.C, PRIORITY<130 ACTION "TACL ALTPRI
<#object>,<#goal>"
```

ASAP Extension

Externally monitored applications cannot meet extreme continuous availability requirements. Even well coded applications that generate alerts at all the proper times get hung; go into processing loops where alerts aren't generated; or are affected by hardware and other external problems. Externally they appear fine, but internally they are not, and the services they provide are either degraded or totally unavailable.

The ASAPX product monitors applications by looking directly inside application programs. By inserting its API procedures into main processing loops ASAPX knows immediately when an error-branch has been taken in the code or when a successful transaction completes. Application service levels can be directly measured, in real-time. Using DOTs goals and actions ASAP users can be alerted when service levels degrade before problems become serious.

Shared memory

ASAPX has no measurable impact on application performance. It accomplishes this by using shared memory in each processor as the primary communication medium between the application and ASAPX. A monitor process in each processor allocates an extensible flat memory segment that is shared by the ASAPX process and by all applications using its API. Application updates to counters and other values are direct memory updates. Other than the initial registration message to the monitor process, there is no inter-process communication.

ASAPX tightly controls shared memory. The ASAPX monitor process and API create and maintain boundary tags and checksums to ensure consistency of all application data allocated in shared memory. Boundary tags are used to uniquely identify an application domain and to protect against cross-boundary move operations. Checksums are computed before and after each memory update to ensure data consistency.

Producing Application Statistics

ASAPX samples application data at each interval by reading its shared-memory segment and performing user-defined computations to produce the values reported for each domain. Data records are generated and written to ASAP Collector processes for storing into the ASAP database.

ASAPX domains are abstract representations of application services. They are long free-form names with up to five levels of hierarchy. For example, an ATM in Chicago might be represented as "Atm\Chicago\$Atm1", "Atm\Chicago\Deposit\$Atm1", or "Deposit\Chicago\$Atm1".

The first (leftmost) level of an ASAPX domain name is the name of an entity as defined in EDL. EDL entities can represent user applications, a subset, or superset of applications, depending on the view the user establishes.

Custom attributes

All of the behavior associated with an entity defined to the ASAPX product, from the way ASAPX defines and treats application data in shared memory to the format of the record produced for the application is defined by EDL. At the entity level, the user defines the contents of shared memory and other entity-wide properties. At the attribute level, the user defines the attributes to be produced, including the formulas used to produce the attribute values at each interval.

Using EDL ASAPX performs custom calculations for each attribute and entity, producing entity-specific records that are sent to the ASAP Collectors for storing in the ASAP database. For example, an entity might define two items in shared memory, a transaction count and an error count. The ASAPX API could be used within the application to update the transaction count after each successful transaction (ENDTRANSACTION) and the error count after each failed transaction (ABORTTRANSACTION).

The two counters could be used to produce a variety of attributes to determine the application service levels. Some of the attributes that could be defined are a transaction count, a transaction rate, an error count, an error rate, a success percentage, a fail percentage, a total transaction count, and a total transaction rate.

ASAPX provides 11 built-in attributes that can be included in an application EDL file. These are automatically computed and formatted by ASAPX when it finds them in application EDL. The ASAPX built-in attributes are:

Avail	Availability of the application domain since it first registered with ASAPX.
Busy	Percent CPU busy for the application process that registered the domain.
CPU	Processor where the registering process executes.
Downtime	Total downtime in seconds since the domain was first registered.
Nak	Number of intervals since application domain was last updated.
Pri	Execution priority of the registering process.
Pstate	Process state of the registering process.
RegTime	Initial date/time domain registered with ASAPX.
Unavail	Unavailability of the domain for the interval.
Version	Version used by the application when registering with ASAPX.
Wstate	Wait state of the registering process.

Aggregation

ASAPX can aggregate data across any level of an application domain name. For example, if application domains used the form "Atm\Chicago\Deposit\$Atm1" then ASAPX can produce an aggregate total record for each of the three upper levels of the name, "Atm", "Atm\Chicago", and "Atm\Chicago\Deposit".

ASAPX automatically propagates the worst state from the aggregate set for each attribute to the aggregate record unless a specific objective has been defined for the attribute at the aggregate level. This gives users the ability to monitor the combined service levels of all application components in a single application view.

Simplicity and ease-of-use

The ASAPX API is simple and easy to understand. There are a total of six API procedures, but only two are required for any application. First, the application calls the ASAP_REGISTER_ procedure to register a domain with ASAPX, and then it calls one of the update procedures, ASAP_UPDATE_ or ASAP_UPDATELIST_, to update one or many of its statistical items in memory at the appropriate points in the application code. The application might also call the ASAP_REMOVE_ procedure to remove itself if operating in batch mode or might remove and re-register if ASAPX returns an error from one of the API calls.

ASAP Hybrid

As discussed in the prior section about the ASAPX Extension, externally monitored applications cannot meet continuous availability requirements. This fact is compounded further when the application to be monitored is running on a remote system without the continuous-availability features of NonStop. Thus a mechanism is needed that permits monitoring these applications from the inside and integrates the metrics and availability statistics gathered from those applications with an availability monitoring infrastructure.

ASAP Hybrid addresses this need for Linux systems by providing an API that allows ASAP to collect metrics directly from a running application. This data is then processed by the ASAP infrastructure in exactly the same manner as all other data: objectives can be set, actions can be taken, data is retained in the historical database, and alerts can be issued through any of the myriad of interfaces provided by ASAP.

Because both ASAP Hybrid and ASAP Extension share nearly identical functional goals, they share many of the same characteristics.

Shared memory

Like ASAP Extension, ASAP Hybrid employs a shared-memory mechanism to collect metrics from an application. This allows the application to simply update values in memory through the ASAP Hybrid API. This API is extremely high-performance and almost completely non-blocking and thus has no measurable impact on applications that utilize it.

There are some technical differences in how the underlying shared memory mechanism works in ASAP Hybrid as compared to ASAP Extension. Key among these are that, instead of using a single large segment shared by all applications, ASAP Hybrid gives each monitored domain its own dedicated smaller segment. This takes advantage of the facilities in Linux that prevent applications from accessing the memory space of another application, thereby yielding the best possible performance.

Support for multi-threaded applications

Linux supports both multi-threaded and single-threaded applications. Multi-threaded applications introduce a requirement to share application metrics between threads of the same process. ASAP Hybrid supports this notion by supplying thread-safe variants of all API calls; these calls regulate access to shared metrics and ensure that multiple threads don't attempt to update the same value at the same time.

While thread-safe variants of API calls are necessary for multi-threaded applications, they have the drawback of introducing potential blocking into the call path. If one thread is updating a shared metric, another thread attempting to update that same metric must wait until the first thread has completed its operation. Single-threaded applications do not need to use a thread-safe API and can avoid the slight penalty of using the thread-safe calls by using the standard API. This gives them the maximum possible performance for their environment with no associated risk.

Platforms

ASAP Hybrid runs on x86 and AMD64 platforms. The ASAP Hybrid API includes both 32-bit and 64-bit libraries, and fully supports both 32-bit and 64-bit applications.

ASAP Hybrid is certified on both Red Hat Enterprise Linux and Suse Linux, but runs on virtually any Linux distribution.

Additional features

As stated previously, ASAP Hybrid and ASAP Extension share nearly identical functional goals. Thus ASAP Hybrid has the same features as ASAP Extension in terms of producing application statistics, defining custom attributes, performing levels of aggregation, and instrumenting applications quickly and easily. See ASAP Extension section for more details on these features.

The ASAP Hybrid API and the ASAPX API are identical in terms of their functionality and nearly identical in terms of procedure names and parameters. There are minor differences in some parameter types due to underlying differences in operating systems. ASAP Hybrid adds thread-safe variants of each API call in order to support multi-threaded applications. Beyond that, there is essentially a one-to-one mapping between ASAP Hybrid API calls and ASAP Extension API calls.

Open Enterprise Management

The HP Open Enterprise Management Gateway (OEM)[14] provides an infrastructure for integrating client monitoring applications, such as ASAP, with Windows-based EMFs. The OEM environment is made up of four distinct pieces, all of which execute on Windows:
1) one or more client applications/agents, such as the ASAP Client
2) the OEM Server
3) one or more EMF Adapters, and
4) one or more EMFs.

Client applications/agents

Client applications/agents serve as data providers to the OEM and integrate directly with OEM Server using the published OEM client APIs. They furnish information on classes, objects, and their associated states to the OEM Server. The OEM is a very general framework and has no specific knowledge of NonStop objects, hierarchy, or naming. Therefore, clients truly can report state information on any application-defined object; clients are not limited by any OEM-imposed view of the object space.

[14] Method and Apparatus for the Open Enterprise Management Gateway (OEM)
Invention disclosure—March 15, 1997—Miller, O'Brien, Tremblay, Boucher

Clients may also optionally define and associate one or more actions with each supplied class or object. These actions are passed through to EMF Adapters and the EMFs themselves, thereby making it possible for users to invoke the actions directly from the OEM or EMF.

Finally, clients also respond to notifications from the OEM Server whenever defined actions are invoked against specific objects. The clients can in turn perform any processing necessary, based upon the action chosen.

OEM Server

The OEM Server acts as a central repository and clearing house of object and state information. It also serves as an intelligent "traffic cop", routing data and requests between client applications and EMF Adapters. The OEM Server provides the following capabilities:

- Maintains an internal database of all defined classes and objects, their states, any details reported by client applications, and all actions defined for those classes/objects.
- Encapsulates the interface to EMFs and isolates client applications from EMFs and EMF behavior. Clients only need to interface with the OEM Server in order to integrate with all supported EMFs. As new EMFs are supported by the OEM, all existing client applications are automatically integrated with that EMF without any code changes to the client.
- Encapsulates the interface to clients and hides implementation details from EMF Adapters. Each EMF Adapter interfaces to the OEM Server and as a result is not required to have specific knowledge of how to interface with any particular client application. This allows users of a particular EMF to invoke client-defined actions against objects without requiring custom logic within the Adapter to deal with each particular client. Therefore, new clients can be developed without having to change existing EMF Adapters.
- Allows integration of EMF state models and client models such as ASAP states: Exists (1), Up (2), Low (3), Medium (4), High (5), Warning (6), Critical (7), Down (8), and Unknown (9).
- Tracks reported state information separately for each client application. This feature, known as state arbitration, makes it possible for multiple clients to report state information on the same object, and ensures that only the most critical information is forwarded to EMFs. For example, if client application A reports that disk $SYSTEM is in state 7, while client application B reports that $SYSTEM is in state 5, the OEM Server will ensure that the more critical state (7) is passed to the EMFs. If application A later reports that the problem with $SYSTEM has been fixed and the state is now 1, the more critical state (5) reported by client B will be forwarded to the EMFs. This differs from the behavior of many EMFs, which do not distinguish between different client applications and always assume the last reported state is the true state of the object.
- Continuously propagates state information throughout its object hierarchy, making it possible to quickly determine the state of higher-level objects by rolling up the states of subordinate objects. Thus users can quickly be directed to domains having problems and can drill down to the specific objects in error.
- Provides standard visualization of all object and state data through Alert windows. Alert windows contain sorted lists of object names and states, from most to least critical. Objects in the most critical states will always "bubble" to the top, making it easy for users to determine which objects are most in need of attention.
- Supports Custom Views of data, which allow users to group heterogeneous objects based upon any criteria they deem appropriate. State values are propagated within the hierarchy of each Custom View independently, so that users can get a high-level picture of the state of business objects and subsystems. For example, a customer might group objects by application—thus a bank could have a Custom View for their ATM application, another for their SWIFT application, and more. The OEM tracks and reports the status of each of these Views independently, allowing customers to track object state by business function rather than from a system standpoint.

EMF Adapters

An EMF adapter is responsible for providing the integration layer between the OEM Server and a single EMF. They interface with the OEM Server through the published OEM adapter APIs and interface with the EMF using whatever scheme is necessary for that EMF. The OEM currently supports four types of adapters: the HP Object Integration Layer (OIL) adapter, which provides a hierarchical tree/Windows Explorer view of objects and states; the CA-TNG Unicenter adapter, which integrates with Computer Associates' TNG framework; and the HP Operations (OpenView) adapter, which integrates with the HP OpenView framework. In addition, customers have developed their own custom adapters using the published OEM adapter API[15].

An EMF adapter encapsulates the interface to a particular EMF and hides all details of that EMF from the OEM Server and client applications. The adapter supplies the EMF with all OEM object, state, and action data by translating standardized OEM requests into EMF-specific operations and functions.

In addition, the adapter allows users to invoke client-defined actions in the EMF and notifies the OEM Server of the invocation by translating EMF-specific operations to standard OEM requests. This in turn allows the OEM Server to notify the necessary client applications that the user has chosen a specific action, and permits the client to respond accordingly.

Enterprise Management Frameworks

Enterprise Management Frameworks play a role within the OEM environment in that they are the ultimate destination for all client-supplied object and state data. As such, they may be true enterprise management frameworks such as HP Operations (OpenView) or CA Unicenter. However, they could just as easily be a user-supplied management application that performs a very specific function. Since EMFs are entirely hidden from applications and the OEM Server, there really are no requirements for what does/does not constitute an EMF. The only true requirement is that an EMF Adapter be supplied that acts as the bridge between the OEM Server and the EMF. Once this piece is in place, the actual EMF can be as complex or simple as necessary based upon the needs of the user.

The OEM Server also provides its own visualization of object and state data through Alert windows and Custom Views. Thus, it can also be used without any EMF whatsoever. The OEM Server, in conjunction with the participating client applications, is capable of supplying both high-level and detail information regarding objects and states, and presents that information to users in a comprehensible and intuitive fashion.

HP Operations (OpenView) Smart Plug-In (SPI) for ASAP

As discussed above, the HP Open Enterprise Management gateway (OEM) allows ASAP integration with frameworks such as HP Operations (OpenView) using a supplied framework adapter. The SPI for ASAP is such an adapter and is intended specifically to share all ASAP object and state data with HP Operations (OpenView).

The OpenView Smart Plug-In for ASAP is included with ASAP for no additional charge. This SPI optionally provides the following features but does not require OpenView to use or deploy ASAP. Thus ASAP can be used with or without OpenView. Features of the SPI for ASAP include:

- ASAP object and state information integrated with HP OpenView screens.
- ASAP alerts appear in OpenView:
  - Availability state info.
  - Performance state info.
  - Service-level objective states.

[15] OEM Annunciator—Copyright 1999-2001 - Rabobank b.v. The Netherlands

- Allows OpenView monitoring of ASAP:
  - System objects such as CPU, Disk, Expand, Comm, Node, System, and Tape
  - Subsystems such as File, Busy, Process, Remote DB, RDF, Spooler, and TMF
  - Application domains such as ATMs, funds, accounts, customers, and orders
  - Third party entities such as ServerNet, and SQL
- Application domains and objects monitored through the ASAPX and ASAP Hybrid are also reported to OpenView automatically. No additional work is necessary to enable this capability. Any object known to ASAP is automatically integrated with OpenView.
- Application objects appear in OpenView the same way as other ASAP objects and can be viewed the same as any ASAP objects, including hierarchical drill down.
- OpenView messages are logged for all ASAP objects that change state. OpenView filtering, forwarding, and reporting tools can be used to notify operations personnel of outages, initiate recovery actions, and more.
- ASAP-generated messages and alerts are also available through the OpenView Web interface. This allows operations personnel to obtain ASAP info from a Web browser.
- Inclusion of ASAP data in OpenView makes heterogeneous system management possible; all HP platforms such as NonStop, Unix, and Windows can be managed from a single console.

For more information

HP NonStop ASAP—Marketing Portal

http://www.hp.com/go/nonstop/ASAP

HP NonStop ASAP—Technical Portal

http://nonstopASAP.com/

HP NonStop ASAP—Manuals

http://docs.hp.com

Technology for better business outcomes

4AA2-5824ENW, April 2009